

Inteligência Computacional

PROJETO – FASE1

Elementos do grupo:

Luís Henrique P. O. Travassos, nº2021136600

Rodrigo Ramalho Ferreira, nº2021139149

Descrição do caso de estudo e objetivos do problema:



- O projeto foca na classificação de imagens de ruas em três categorias: limpas, sujas e com ecopontos, com o objetivo de promover a conscientização sobre a limpeza urbana.
- O objetivo é desenvolver um modelo de machine learning preciso para monitorar o estado das ruas. Serão usadas 5400 fotos, perto de 1800 para cada classe, para treinar o modelo.
- O projeto prevê ajustes na arquitetura da rede neural e nos parâmetros de treinamento, assim como a aplicação de técnicas de pré-processamento de imagens.
- Como visão futura, este modelo poderia ser aplicado em um contexto mais amplo de gestão sustentável urbana, contribuindo para a melhoria do ambiente urbano.

Descrição da implementação dos algoritmos:

- Modelo de Dados MLP
 - Uso de tensorflow para criar um sistema de classificação de imagens com redes neuronais.
- Importação de Bibliotecas:
 - Importação das bibliotecas essenciais.
- Definição de Parâmetros:
 - Diretório de Dados: 'Dataset2/'
 - Tamanho das Imagens: 350 pixels
 - Tipos de Ruas: limpas, sujas, com ecopontos
- Organização dos Dados:
 - Criação de DataFrame para os nomes de arquivos.
- Verificação do Dataset:
 - Resumo estatístico da distribuição de imagens.
- Carregamento e Pré-processamento:
 - Conversão em arrays numpy
 - Normalização dos pixels
- Preparação dos Dados:
 - Separação de treinamento e teste.
- Definição, Treinamento e Avaliação do Modelo:
 - Rede neural com otimizador Adam.
- Avaliação de Desempenho:
 - Métricas: matriz de confusão, relatório de classificação e AUC.

```
# Importando as bibliotecas necessárias
from sklearn.preprocessing import LabelEncoder, OneHotEncoder # Para pré-processamento de dados
from sklearn.utils import shuffle # Para embaralhar os dados
from sklearn.model_selection import train_test_split # Para dividir o conjunto de dados em treinamento e teste
from sklearn.metrics import ( # Métricas de avaliação de modelo
    confusion_matrix, accuracy_score, recall_score, f1_score, roc_auc_score, classification_report
)

from tensorflow.keras.optimizers import Adam # Otimizador para a rede neural
from tensorflow.keras.preprocessing.image import load_img, img_to_array # Pré-processamento de imagens

import keras # Biblioteca Keras para criação de modelos de redes neurais
import tensorflow as tf # TensorFlow para construção de redes neurais

import matplotlib.pyplot as plt # Para plotagem de gráficos
import pandas as pd # Manipulação de dados em formato tabular
import numpy as np # Biblioteca Numpy para manipulação de matrizes
import cv2 # OpenCV para processamento de imagens
import os # Operações com sistema de arquivos
```

```
# path = 'Dataset2/' # Definindo o caminho do diretório contendo o dataset
# im_size = 350 # Definindo o tamanho desejado para as imagens

street_types = os.listdir(path) # Listando os tipos de ruas disponíveis no diretório
print(street_types) # Imprimindo os tipos de ruas encontrados
print("Tipos de ruas encontrados: ", len(street_types)) # Imprimindo o número de tipos de ruas encontrados

['clean', 'litter', 'recycle']
Tipos de ruas encontrados: 3
```

```
# streets = [] # Lista vazia para armazenar informações das ruas

# Iterando sobre os diferentes tipos de ruas
for item in street_types:
    all_streets = os.listdir(path + '/' + item) # Obtendo todos os nomes de arquivo na pasta

    # Adicionando os nomes de arquivo à lista 'streets'
    for street in all_streets:
        streets.append(item, str(path + '/' + item) + '/' + street)

# Criando um DataFrame da pandas com as informações das ruas
streets_df = pd.DataFrame(data=streets, columns=['street type', 'image'])
```

```
# Imprimindo o número total de ruas no conjunto de dados
print("Total number of streets in the Dataset: ", len(streets_df))

# Contando o número de ruas em cada categoria e imprimindo os resultados
street_count = streets_df['street type'].value_counts()
print("Ruas em cada categoria: ")
print(street_count)

Total number of streets in the Dataset: 5405
Ruas em cada categoria:
street type
recycle    1875
clean     1825
litter     1705
Name: count, dtype: int64
```

```
# images = [] # Lista para armazenar as imagens
# labels = [] # Lista para armazenar as etiquetas das ruas

# Iterando sobre os diferentes tipos de ruas
for label in street_types:
    data_path = os.path.join(path, label) # Obtendo o caminho para os dados de uma categoria
    filenames = os.listdir(data_path) # Obtendo os nomes dos arquivos de imagem na categoria

    # Iterando sobre os arquivos de imagem na categoria
    for filename in filenames:
        img_path = os.path.join(data_path, filename) # Obtendo o caminho completo da imagem
        img = load_img(img_path, target_size=(im_size, im_size)) # Carregando a imagem e redimensionando
        img_array = img_to_array(img) # Convertendo a imagem para um array numpy
        images.append(img_array) # Adicionando a imagem à lista de imagens
        labels.append(label) # Adicionando a etiqueta da rua à lista de etiquetas

# Convertendo a lista de imagens para um array numpy e normalizando os valores de pixel
images = np.array(images)
images = images.astype('float32') / 255.0

# Obtendo a forma (shape) do array de imagens
images.shape

[5405, 350, 350, 3]
```

```
# Obtendo as etiquetas das ruas do DataFrame
y = streets_df['street type'].values

# Convertendo as etiquetas para valores numéricos usando LabelEncoder
y_labelencoder = LabelEncoder()
y = y_labelencoder.fit_transform(y)

# Embaralhando as imagens e suas etiquetas para garantir uma mistura aleatória
images, y = shuffle(images, y, random_state=1)

# Dividindo os dados em conjuntos de treinamento e teste
train_x, test_x, train_y, test_y = train_test_split(images, y, test_size=0.05, random_state=415)

# Imprimindo as formas (shapes) dos conjuntos de dados
print(train_x.shape)
print(test_x.shape)
print(train_y.shape)
print(test_y.shape)

(5134, 350, 350, 3)
(271, 350, 350, 3)
(5134,)
(271,)
```

```
# Definindo o modelo da rede neural
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(im_size, im_size, 3)), # Camada de achatamento para transformar a imagem em vetor
    keras.layers.Dense(256, activation=tf.nn.tanh), # Camada densa com ativação tangente hiperbólica
    keras.layers.Dense(128, activation=tf.nn.relu), # Camada densa adicional com ativação ReLU
    keras.layers.Dense(64, activation=tf.nn.relu), # Outra camada densa adicional com ativação ReLU
    keras.layers.Dense(3, activation=tf.nn.softmax) # Camada de saída com ativação softmax (3 classes)
])

# Compilando o modelo
model.compile(optimizer=Adam(learning_rate=0.01),
              loss='sparse_categorical_crossentropy', # Função de perda para classificação multiclasse
              metrics=['accuracy']) # Métrica de avaliação

# Treinando o modelo
model.fit(train_x,
        train_y,
        epochs=10, # Número de épocas de treinamento
        batch_size=64, # Tamanho do lote
        validation_data=(test_x, test_y)) # Dados de validação

# Obtendo as probabilidades de classe previstas
y_probs = model.predict(test_x)

# Obtendo as previsões finais (índices da classe com a maior probabilidade)
y_pred = np.argmax(y_probs, axis=-1)
```

```
# Calculando a matriz de confusão
confusion = confusion_matrix(test_y, y_pred)
print("Matriz de Confusão:")
print(confusion)




# Gerando um relatório de classificação
report = classification_report(test_y, y_pred, target_names=street_types, zero_division=1)
print(report)

# Calculando a Área sob a Curva (AUC) para cada classe
for i in range(len(street_types)):
    auc = roc_auc_score(test_y == i, y_probs[:, i])
    print(f"AUC para classe {street_types[i]}: {auc}")
```

Análise de resultados:

APRESENTAÇÃO DATASETS (Dataset Normal)

O conjunto de dados original para este problema consiste em 5405 fotos, todas com dimensões variadas. É necessário redimensioná-las no código para garantir uniformidade. Cada uma das três classes - ruas limpas, sujas e com ecopontos - contém aproximadamente 1800 fotos, todas apresentando certo nível de complexidade.

Rua Limpa	Rua Suja	Rua com Ecopontos
		

Análise de resultados:

Dataset normal com modelo de dados MLP (melhor resultado)

Configuração

```
# Definindo o modelo da rede neural
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(im_size, im_size, 3)),
    keras.layers.Dense(1024, activation=tf.nn.relu),
    keras.layers.Dense(512, activation=tf.nn.relu),
    keras.layers.Dense(256, activation=tf.nn.relu),
    #keras.layers.Dense(128, activation=tf.nn.relu),
    #keras.layers.Dense(64, activation=tf.nn.relu),
    #keras.layers.Dense(32, activation=tf.nn.relu),
    keras.layers.Dense(3, activation=tf.nn.softmax)
])

# Compilando o modelo
model.compile(optimizer=Adam(learning_rate=0.009),
              loss='sparse_categorical_crossentropy', # Função de perda
              metrics=['accuracy']) # Métrica de avaliação
```

Resultados

Matriz de Confusão:

```
[[75 19  4]
 [15 53  6]
 [10 15 74]]
```

	precision	recall	f1-score	support
clean	0.75	0.77	0.76	98
litter	0.61	0.72	0.66	74
recycle	0.88	0.75	0.81	99
accuracy			0.75	271
macro avg	0.75	0.74	0.74	271
weighted avg	0.76	0.75	0.75	271

AUC para classe clean: 0.8976052848885219

AUC para classe litter: 0.8656880230484291

AUC para classe recycle: 0.9168428470754052

- Analisando estes resultados, é possível verificar que o desempenho do modelo é consideravelmente satisfatório, mas longe de excelente, tendo em conta que a complexidade do conjunto de dados é alta. De seguida iremos analisar os outros casos.

Análise de resultados:

Dataset normal com modelo de dados MLP (mais camadas)

Configuração

```
# Definindo o modelo da rede neural
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(im_size, im_size, 3)),
    keras.layers.Dense(1024, activation=tf.nn.relu),
    keras.layers.Dense(512, activation=tf.nn.relu),
    keras.layers.Dense(256, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    #keras.layers.Dense(64, activation=tf.nn.relu),
    #keras.layers.Dense(32, activation=tf.nn.relu),
    keras.layers.Dense(3, activation=tf.nn.softmax)
])

# Compilando o modelo
model.compile(optimizer=Adam(learning_rate=0.009),
              loss='sparse_categorical_crossentropy', # Função de p
              metrics=['accuracy']) # Métrica de avaliação
```

Resultados

```
Matriz de Confusão:
[[47 48  3]
 [ 9 55 10]
 [ 3 32 64]]

      precision    recall  f1-score   support

   clean         0.80         0.48         0.60         98
   litter         0.41         0.74         0.53         74
  recycle         0.83         0.65         0.73         99

 accuracy                   0.61         271
 macro avg         0.68         0.62         0.62         271
 weighted avg         0.70         0.61         0.63         271

AUC para classe clean: 0.8589123510675946
AUC para classe litter: 0.7231444642612156
AUC para classe recycle: 0.872093023255814
```

- Para esta análise iremos alterar o modelo de dados MLP de forma a ele possuir mais camadas, sem afetar muito o número de neurónios, para observar que tipo de resultados podemos obter.
- Analisando estes resultados, a configuração com mais camadas não parece ser uma melhoria em relação à melhor configuração. Parece que, ao adicionar mais camadas, o modelo pode ter se tornado mais complexo e pode estar sofrendo de overfitting ou dificuldades em generalizar para novos dados.
- A configuração com mais camadas teve uma melhora na precisão da classe "clean", mas um desempenho significativamente pior para as classes "litter" e "recycle". A precisão geral do modelo (acurácia) também diminuiu, indicando que a configuração com mais camadas não é tão eficaz em geral.

Análise de resultados:

Dataset normal com modelo de dados MLP (menos camadas)

Configuração

```
# Definindo o modelo da rede neural
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(im_size, im_size, 3)),
    keras.layers.Dense(1024, activation=tf.nn.relu),
    keras.layers.Dense(512, activation=tf.nn.relu),
    #keras.layers.Dense(256, activation=tf.nn.relu),
    #keras.layers.Dense(128, activation=tf.nn.relu),
    #keras.layers.Dense(64, activation=tf.nn.relu),
    #keras.layers.Dense(32, activation=tf.nn.relu),
    keras.layers.Dense(3, activation=tf.nn.softmax)
])

# Compilando o modelo
model.compile(optimizer=Adam(learning_rate=0.009),
              loss='sparse_categorical_crossentropy', # Função de
              metrics=['accuracy']) # Métrica de avaliação
```

Resultados

Matriz de Confusão:

[[98 0 0]				
[74 0 0]				
[99 0 0]]				
	precision	recall	f1-score	support
clean	0.36	1.00	0.53	98
litter	1.00	0.00	0.00	74
recycle	1.00	0.00	0.00	99
accuracy			0.36	271
macro avg	0.79	0.33	0.18	271
weighted avg	0.77	0.36	0.19	271

AUC para classe clean: 0.5037454288073611
AUC para classe litter: 0.4915283303608177
AUC para classe recycle: 0.5035236081747709

- Para esta análise iremos alterar o modelo de dados MLP de forma a ele possuir menos camadas, sem afetar muito o número de neurónios, para observar que tipo de resultados podemos obter.
- Relativamente a estes resultados, a configuração com menos camadas apresenta um desempenho muito pior em comparação com a melhor configuração. Na verdade, não está conseguindo aprender corretamente as características das classes "litter" e "recycle", resultando em uma classificação praticamente ineficaz.

Análise de resultados:

Dataset normal com modelo de dados MLP (mais neurónios)

Configuração

```
# Definindo o modelo da rede neural
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(im_size, im_size, 3)),
    keras.layers.Dense(2048, activation=tf.nn.relu),
    keras.layers.Dense(512, activation=tf.nn.relu),
    keras.layers.Dense(256, activation=tf.nn.relu),
    #keras.layers.Dense(128, activation=tf.nn.relu),
    #keras.layers.Dense(64, activation=tf.nn.relu),
    #keras.layers.Dense(32, activation=tf.nn.relu),
    keras.layers.Dense(3, activation=tf.nn.softmax)
])

# Compilando o modelo
model.compile(optimizer=Adam(learning_rate=0.009),
              loss='sparse_categorical_crossentropy', # Função
              metrics=['accuracy']) # Métrica de avaliação
```

Resultados

Matriz de Confusão:

```
[[98  0  0]
 [74  0  0]
 [99  0  0]]
```

	precision	recall	f1-score	support
clean	0.36	1.00	0.53	98
litter	1.00	0.00	0.00	74
recycle	1.00	0.00	0.00	99
accuracy			0.36	271
macro avg	0.79	0.33	0.18	271
weighted avg	0.77	0.36	0.19	271

AUC para classe clean: 0.5037454288073611
AUC para classe litter: 0.4915283303608177
AUC para classe recycle: 0.5035236081747709

- Para esta análise iremos alterar o modelo de dados MLP de forma a ele possuir mais neurónios, sem afetar o número de camadas, para observar que tipo de resultados podemos obter.
- Relativamente a estes resultados parece que simplesmente aumentar o número de neurónios não está sendo eficaz para melhorar o desempenho do modelo.

Análise de resultados:

Dataset normal com modelo de dados MLP (menos neurónios)

Configuração

```
# Definindo o modelo da rede neural
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(im_size, im_size, 3)),
    keras.layers.Dense(512, activation=tf.nn.relu),
    keras.layers.Dense(256, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    #keras.layers.Dense(128, activation=tf.nn.relu),
    #keras.layers.Dense(64, activation=tf.nn.relu),
    #keras.layers.Dense(32, activation=tf.nn.relu),
    keras.layers.Dense(3, activation=tf.nn.softmax)
])

# Compilando o modelo
model.compile(optimizer=Adam(learning_rate=0.009),
              loss='sparse_categorical_crossentropy', # Função de perda
              metrics=['accuracy']) # Métrica de avaliação
```

Resultados

Matriz de Confusão:

```
[[98  0  0]
 [74  0  0]
 [99  0  0]]
```

	precision	recall	f1-score	support
clean	0.36	1.00	0.53	98
litter	1.00	0.00	0.00	74
recycle	1.00	0.00	0.00	99
accuracy			0.36	271
macro avg	0.79	0.33	0.18	271
weighted avg	0.77	0.36	0.19	271


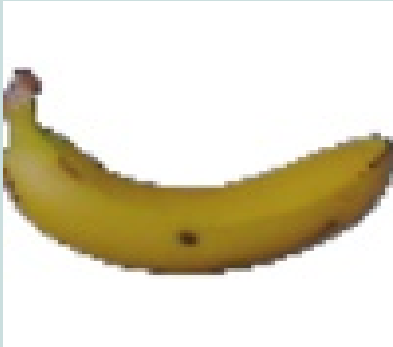

```
AUC para classe clean: 0.5037454288073611
AUC para classe litter: 0.4915283303608177
AUC para classe recycle: 0.5035236081747709
```

- Para esta análise iremos alterar o modelo de dados MLP de forma a ele possuir menos neurónios, sem afetar o número de camadas, para observar que tipo de resultados podemos obter.
- Relativamente a estes resultados parece que ajustar o número de neurónios não está produzindo melhorias significativas no desempenho do modelo.

Análise de resultados:

APRESENTAÇÃO DATASETS (Dataset Menor)

O dataset secundário possui imagens mais simples e claras em comparação com o conjunto principal. Isso facilita a análise do código, pois o número de exemplos e classes foi consideravelmente reduzido, permitindo avaliar a eficiência da implementação sem lidar com um grande volume de dados. Este conjunto menor inclui 1472 fotos, todas com dimensões idênticas. Cada uma das três categorias - maçãs, bananas e cocos - tem cerca de 490 imagens, todas de baixa complexidade.

Maça	Banana	Coco
		

Análise de resultados:

Dataset menor com modelo de dados MLP (melhor resultado)

Configuração

```
# Definindo o modelo da rede neural
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(im_size, im_size, 3)),
    keras.layers.Dense(1024, activation=tf.nn.relu),
    keras.layers.Dense(512, activation=tf.nn.relu),
    keras.layers.Dense(256, activation=tf.nn.relu),
    #keras.layers.Dense(128, activation=tf.nn.relu),
    #keras.layers.Dense(64, activation=tf.nn.relu),
    #keras.layers.Dense(32, activation=tf.nn.relu),
    keras.layers.Dense(3, activation=tf.nn.softmax)
])

# Compilando o modelo
model.compile(optimizer=Adam(learning_rate=0.009),
              loss='sparse_categorical_crossentropy', # Função de perda
              metrics=['accuracy']) # Métrica de avaliação
```

Resultados

```
Matriz de Confusão:
[[30  0  0]
 [ 0 24  0]
 [ 0  0 20]]
```

	precision	recall	f1-score	support
Apple	1.00	1.00	1.00	30
Banana	1.00	1.00	1.00	24
Cocos	1.00	1.00	1.00	20
accuracy			1.00	74
macro avg	1.00	1.00	1.00	74
weighted avg	1.00	1.00	1.00	74

```
AUC para classe Apple: 1.0
AUC para classe Banana: 1.0
AUC para classe Cocos: 1.0
```

- O objetivo desta análise é determinar se o desempenho não excelente no conjunto de dados principal está relacionado à complexidade dos dados ou se existem outros fatores a serem considerados na melhoria da implementação do modelo.
- Esta configuração com um dataset mais simples e menor demonstra que o desempenho do modelo está diretamente relacionado à qualidade e complexidade do dataset.

Conclusões:

Ao analisar os diferentes testes com variações nas configurações do modelo de MLP, podemos fazer as seguintes conclusões gerais:

- **Variação nas Camadas:** Adicionar mais ou menos camadas à rede neural não resultou em melhorias significativas. Na verdade, em alguns casos, houve uma piora no desempenho, indicando possíveis problemas de overfitting.
- **Variação no Número de Neurônios:** Aumentar ou diminuir o número de neurônios também não teve um impacto discernível no desempenho do modelo. Isso sugere que a complexidade da arquitetura da rede não estava diretamente correlacionada com o desempenho.
- **Variação no Dataset:** A configuração com um dataset mais simples e menor mostrou resultados excepcionais, com todas as métricas indicando um desempenho perfeito. Isso destaca a importância crítica da qualidade e complexidade do dataset no sucesso do modelo.

Em resumo, o desempenho de um modelo de rede neural está intrinsecamente ligado à qualidade e complexidade do dataset, bem como à seleção adequada de arquitetura e hiperparâmetros e experimentar diferentes configurações e técnicas de melhoria pode levar a um modelo mais eficaz e preciso.

Referências

- Pesquisas de Dataset's:
 - <https://kaggle.com/>
 - <https://archive.ics.uci.edu/datasets>
 - <https://paperswithcode.com/datasets>
- Ferramenta de criação do Dataset:
 - [https://chrome.google.com/webstore/search/all image downloader?hl=pt-PT](https://chrome.google.com/webstore/search/all+image+downloader?hl=pt-PT)
- Bibliotecas:
 - <https://scikit-learn.org/stable/>
 - <https://www.tensorflow.org/>
 - <https://keras.io/>